TDB-ACC-NO: NN8006341

DISCLOSURE TITLE: Macro Physical-To-Logical Checking LSI Chip Design. June 1980.

PUBLICATION-DATA: IBM Technical Disclosure Bulletin, June 1980, US

VOLUME NUMBER: 23

ISSUE NUMBER: 1

PAGE NUMBER: 341 - 345

PUBLICATION-DATE: June 1, 1980 (19800601)

CROSS REFERENCE: 0018-8689-23-1-341

DISCLOSURE TEXT:

5p. There is an increasing trend towards macro-based design of
LSI chips, and it is becoming essential to perform the physical to
logical check at the macro level. This article describes a method
for performing the macro-level check.
- The drawing is an example of a macro with external pins, A1,
A2, B1, B2, C1 and C2. Such macros will be interconnected by nets to
perform the desired logic/analog function. To illustrate a
fundamental problem encountered in physical to logical checking at
the macro level, let us consider the case where the example macro
contains two AND circuits. Inputs A1 and A2 are 'ANDed' to produce
output C1, and inputs B1 and B2 are 'ANDed' to produce output C2.
Clearly, inputs A1 and A2 are swappable, i.e., interchangeable, as
are inputs B1 and B2. Outputs C1 and C2 however can

only be swapped
if the corresponding input pairs are swapped. These possibilities
are characterized for the macro by the following SWOP rule:
SWOP = ((A1<>A2),C1)<>((B1<>B2),C2): where the symbol '<>' represents
'can be swopped with'.
- SWOP Rule notation definition:
The notation of the SWOP rule is defined in the meta language of SL1
by the following production rules:

```
swop ->,.swopel ...
swopel -> <> .  (swop)  ...  Ý <>.integer ...
```

where the symbol '<>' means that the preceding swop element is
swoppable with the succeeding swop element.
- In order to implement the swop rule in the machine, let us
define an array S(PN, O:GN) CHAR (1), where GN is the number of nest
levels in the SWOP rule.
- PN is the number of pins in the SWOP rule.
- In practice take S(1:150, 0:10) and define
#G(fixed bin (31)) to be the maximum nest depth.
(max nest depth is closest to pins).
and #P(fixed bin(31)) to be the number of pins.
- Using the defined swoppability rule syntax, let us define the
example:

```
SWOP = ((A,B)<>(C,D))<>((E,F)<>(G,H),I,K,L<>M,N).
```

This is processed to produce the following assignments to S.
(see original)
S Array Optimization:
At parse time, the swop rule is read into a character string, and
redundant parentheses are removed using the rules:
1. A pair of corresponding '0' are redundant if they are preceded
and followed by parentheses. i.e., ((A,B)) equivalent to (A,B)
2. A pair of corresponding '0' are redundant if they are preceded
and followed by commas.
i.e. ,(A,B), equivalent to A,B
,(A<>B), equivalent to A<>B
3. The pair of corresponding '0', which contain the

whole swop rule,
are redundant if coded.
i.e., SWOP = (A,B,C<>D) equivalent to A,B,C<>D
4.
Special Case - first and/or last items '0' are redundant if
preceded by ',' and followed by nothing or preceded by nothing and
followed by ','.
i.e., (A),B,C,D equivalent to A,B,C,D
A,B,C,(D) equivalent to A,B,C,D.
All four rules are applied repeatedly until no further reduction is
made. At this point the S array is built.
- Subroutine MACHEK:
The purpose of this routine is to check whether the list of physical
pin names and associated net names, previously set up in the data
base (ADB), are functionally equivalent to the list of logical pin
names and associated net names, read from the logical description,
taking into account the fact that pins may be swopped within certain
groups as defined in the logical description and groups may be
swopped within 'super groups', etc.
- The first call to the program is an initial call which reads in
the complete physical list of pin names and net names from the ADB.
This list is in pin name order.
- Subsequent, normal calls to this program give it the lists of
logical pin names and net names for a particular macro circuit
invocation together with a table specifying the extent to which pins
and groups of pins, etc., can be swopped for the macro while still
performing the same circuit function.
- A binary search is made in the list of physical pin names for
the first logical pin name, and the corresponding physical net name
is saved in a work area. Sequential searches are made, backwards or

forwards, from this point, to find the remaining physical net names
corresponding to the remaining logical pin names.

- Clearly, if the pin names cannot be found, then we immediately
have failure to match. Otherwise, we have two lists of net names and
should be able to rearrange the physical ones in accordance with the
swop data to match the logical ones. The swop data is transposed to
a form suitable for the method used. In the first column, each row
represents the swoppability of each pin with the following pin 1 if
yes, 0 if no.

- For the next column, a 1 signifies that this pin is the first
of a group of pins which can be swopped with the following group, a 0
signifies that this pin is the first of a group which cannot be
swopped, and a) signifies the last pin of a group. For the next
column the same values indicate the swoppability of groups of groups,
and so on.

- An internal recursive subroutine MATCH is used to check whether
the lists of net names correspond or can be swopped to do so. It
works by descending recursively to lower and lower levels until it is
checking pins within a group. It is started off looking at the
highest level of grouping and at the first net names in each list.
At this level, its result, success or failure, is the overall result
and is returned by MACHEK.

- MATCH works in two different ways depending on whether it is
working at the lowest level (checking pins within a group) or not
(checking groups within super groups, etc.).

- At the lowest level, MATCH works row by row down the two lists,
comparing net names. If the two members for a row are

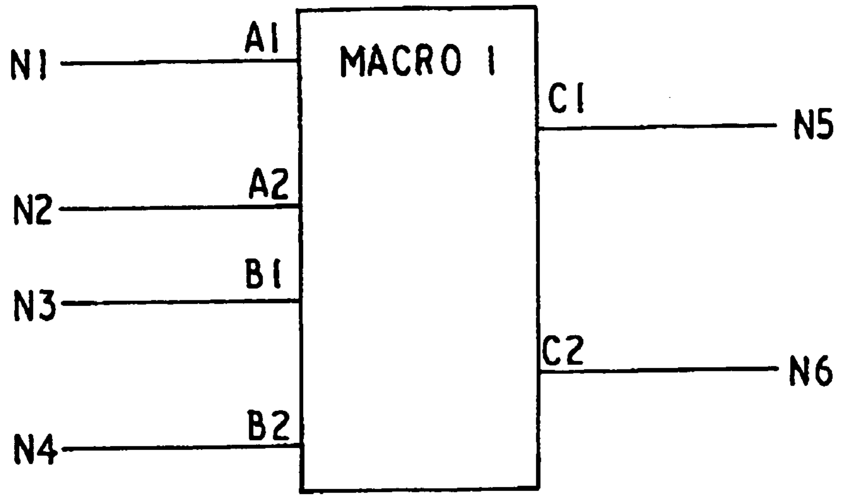
unequal the
    swop data for the first column of this row is examined
to see whether
    the following physical net name can be tried instead.
If so, and if
    it matches, then it is swopped, and we proceed to the
next row. If
    not, then we have failed at this level, and MATCH
returns this result
    to its calling higher level. Note that if such a
failure occurs and
    we have already had some matching rows at this level,
then there is
    an overall failure.
    -       At other levels MATCH works by setting up a
recursive call to
    itself at the next lower level. For example, consider
a call at
    level two, i.e., checking groups within super groups.
Here MATCH
    sets up a call to itself at level one (i.e., checking
pins within a
    group) for the first groups within the super groups it
is given. If
    this returns a success then this level advances to the
next groups
    within the given super groups. If level one returns a
failure, then
    MATCH examines the swop data to see whether the group
can be swopped
    with the next within the super group. If so, another
call is made,
    and so on. If success occurs, the groups are swopped
and the program
    advances; otherwise, failure is returned to the higher
calling level,
    and so on.
    -       This process continues until the final result
is returned to
    the highest level. Note, as before, that matching
failure following
    partial success at any level implies an immediate
overall failure;
    complete success takes longer to establish]

SECURITY: Use, copying and distribution of this data is
subject to the
restictions in the Agreement For IBM TDB Database and

Related Computer
Databases. Unpublished - all rights reserved under the
Copyright Laws of the
United States. Contains confidential commercial information
of IBM exempt
from FOIA disclosure per 5 U.S.C. 552(b)(4) and protected
under the Trade
Secrets Act, 18 U.S.C. 1905.

COPYRIGHT STATEMENT: The text of this article is
Copyrighted (c) IBM
Corporation 1980. All rights reserved.

MACRO CHECKING EXAMPLE



```
         2        1        0
S====>+___________________________+                 +___+  <====PNAME
      |                            |                |     |
      |  1        1        0       |                |  A  |
      |  )                 0       |                |  B  |
      |  0                 0       |                |  C  |
      |  )        )        0       |                |  D  |     For this example #G=2
      |  1        0        0       |                |  E  |
      |  )                 0       |                |  F  |
      |  0                 0       |                |  G  |
      |  )        )        0       |                |  H  |
      |                    0       |                |  I  |
      |                    0       |                |  K  |
      |                    1       |                |  L  |
      |                    0       |                |  M  |
      |                    0       |                |  N  |
      +___________________________+                 +___+

         A        A        A
         |        |        |
         |        |        +____  Defines Pin swoppability.
         |        |               '1' implies pin can be swopped with next pin
         |        |               '0' implies that it cannot.
         |        |
         |        +_____________  Defines Group 1 swoppability.
         |                        '1' implies group can be swopped with next
         |                            group.
         |                        '0' implies that it cannot.
         |
         +______________________  Defines Group 2 swoppability.
                                  ditto Group 1.
NB. ')' denotes end of group.
```